This paper has been accepted for publication by IEEE under the copyright notion of 978-1-5090-5049-9/17/$31.00 © 2017 IEEE.
The final authenticated version is available at: https://10.1109/BlackSeaCom.2017.8277685

ELATS: Energy and Locality Aware Aggregation Tree for Skip Graph

Yahya Hassanzadeh-Nazarabadi and Öznur Özkasap
Department of Computer Engineering, Koç University, İstanbul, Turkey
{yhassanzadeh13, oozkasap}@ku.edu.tr

***Abstract*—As a distributed hash table (DHT), Skip Graph acts as the underlying routing infrastructure of peer-to-peer (P2P) storage systems, distributed online social networks, search engines, and other DHT-based applications. For many P2P applications, data aggregation is vital, however, it is a missing feature of Skip Graph. The traditional aggregation algorithms cost noticeable message overhead which degrades the energy efficiency while increasing the response time. Likewise, the aggregation trees proposed for other DHTs are either inapplicable to the Skip Graph or apply some sort of randomness in their construction. Randomized features of an aggregation tree result in higher aggregation latency as well as enforcing unbalanced load on nodes which negatively affect the energy efficiency. In this paper, we propose *ELATS* which is the first energy and locality aware aggregation tree for Skip Graph. We define the energy awareness as minimizing the average energy cost of an aggregation tree, and the locality awareness as minimizing the latency on the path between the root and leaves of the aggregation tree. Performance analysis results show that *ELATS* algorithm provides both energy and locality awareness, and improves the aggregation latency with the gain of about 8% in comparison to the best existing solutions for DHTs which are either locality aware or energy aware.**

I. Introduction

Skip Graph [1] is a prefix-based routing DHT where nodes are connected to each other based on their identifiers' common prefixes. Fast searching [1], low traffic [2], high guarantee [2], scalability, and load balancing features of Skip Graph make it a suitable routing approach for P2P cloud services [3]–[8]. Likewise, Skip Graph can be used as an efficient underlying structure in applications ranging from distributed storage systems [9], to online social networks [10], and search engines [11].

Nodes in P2P systems employ data aggregation to obtain an identical view about a certain parameter of the system which is referred as the **parameter of interest**. Examples for parameter of interest in P2P systems are size of system [12], average available bandwidth, the reputation of a data object [13], and local clock [14]. Data aggregation is defined as repeatedly obtaining data about the parameter of interest from several peers, compressing or reducing the size of obtained data to achieve a smaller sized **aggregated data**, and forwarding the aggregated data to other peers of the system which causes all peers converge to a similar or identical view of the parameter of interest. Skip Graph suffers from the lack of a data aggregation capability. The traditional distributed data aggregation solutions are based on forwarding aggregated data to a subset of nodes (i.e. gossiping [15]–[17]), passing the aggregated data one by one (i.e. random walk [18]–[20]), and forwarding the aggregated data to all nodes (i.e. flooding). The traditional solutions, with the drawback of high message overhead, result in high aggregation latency and response time. Likewise, discarding the constraint on the number of exchanged messages as well as the size of the messages, the traditional solutions cause additional energy cost for the system.

The existing DHT-based solutions aim at constructing an aggregation tree on top of the underlying routing infrastructure, where the aggregation is done in two phases: request and reply. In the **request phase**, the root node initiates the aggregation request message and sends it to its children. The aggregation request message is forwarded in a top-down manner by each node to its corresponding children set until it reaches the leaves. Once a leaf node receives the aggregation request, it starts the **reply phase** by sending its state of the parameter of interest to its parent. Each node combines its parameter's state with all aggregated states obtained from its children and sends it to its parent. The reply phase continues in a bottom-up manner until it reaches the root node. The root node generates the final result of aggregation by combining the aggregated states obtained from its children. If all nodes need to be notified about the aggregation result, the root node sends the result in a top-down manner similar to the request phase.

The existing DHT-based aggregation trees are constructed based on the specific features of the system. For example [21] proposes an aggregation tree for Chord based on its circular property. So far, no aggregation tree is constructed specifically for Skip Graph. Likewise, the existing aggregation tree schemes for other prefix-based DHTs mainly focused on response time and does not consider the energy cost enforced by aggregation tree to the peers.

In this study, we propose **ELATS, the first energy and locality aware aggregation tree for Skip Graph**. We define the locality awareness of an aggregation tree as minimizing the latency on the path between the root and the leaves. This improves the performance of aggregation tree in term of response time. We define the energy awareness as minimizing the average energy cost of the non-leaf nodes in the aggregation tree. The original contributions of this paper are as follows:

- We propose ELATS which is the first fully decentralized energy and locality aware aggregation tree approach for the Skip Graph. ELATS is a fully decentralized recursive algorithm with each recursion executed on a single node.

- We define the energy cost model of a generic aggregation tree, and extend the Skip Graph simulator SkipSim [22] to support the aggregation tree models.
- We redesign the proposed aggregation tree algorithms for other DHTs to be applicable on the Skip Graph, model them in the SkipSim and compare their performance with ELATS.
- Our simulation results show that compared to the best existing solutions which are either locality aware or energy aware, *ELATS* provides both the energy and locality awareness, and improves the aggregation latency with the gains of about **8%**.

II. Aggregation Tree: Energy Cost Model

In our view of an aggregation tree, a node consumes energy while it exchanges request/reply messages, or aggregates the obtained data from its children [23]. All the aggregated data have the identical size which implies identical size of all the exchanged messages. We assume that exchanging a single message between two nodes in the aggregation tree charges an energy cost of E units on each of them.

Energy Cost of Request Phase: The root node r initiates request phase by sending an aggregation request to its children. Equation 1 indicates the energy cost of the root node in the request phase, where the children set of r is denoted by CS_r.

$$E_{request}(r) = |CS_r| \times E \tag{1}$$

During the request phase, an intermediate node i receives the request message from its parent, and forwards the request to its children. Equation 2 shows the energy cost of intermediate node i during the request phase.

$$E_{request}(i) = E + |CS_i| \times E \tag{2}$$

Equation 3 represents the average energy cost of non-leaf nodes in the aggregation tree T during the request phase, denoted by $E_{request}$. I_T corresponds to the set of all intermediate nodes of T.

$$E_{request} = \frac{E_{request}(r) + \sum_{i \in I_T} E_{request}(i)}{|I_T| + 1} \tag{3}$$

Energy Cost of Reply Phase: During the reply phase, an intermediate node i receives the aggregated states of nodes covered by sub-tree rooted at node i, adds its state of the parameter of interest to the set of obtained aggregated states, combines the states in the set, and sends the combined aggregated states to its parent. Equation 4 shows the energy cost of an intermediate node i during the reply phase. U denotes the energy cost of combining one pair of aggregated states. We assume that the combination is done linearly. The first two aggregated states are combined, and the result is iteratively combined with the next aggregated states until no aggregated state remains. Doing so, combining $|CS_i|$ aggregated states as well as node's i state of parameter of interest is done in $|CS_i|$ steps, which costs the energy cost of $|CS_i| \times U$.

$$E_{reply}(i) = (|CS_i| \times E) + (|CS_i| \times U) + E \tag{4}$$

In the last step of the reply phase, the root node, r, receives the aggregated states of all nodes in the subtrees rooted at its children, adds its state of the parameter of interest to the set of obtained aggregated states, and combines the aggregated states. This results in the aggregated state of all nodes in the aggregation tree. Equation 5 shows the energy cost of the root node during the reply phase.

$$E_{reply}(r) = (|CS_r| \times E) + (|CS_r| \times U) \tag{5}$$

Equation 6 shows the average energy cost of non-leaf nodes in the aggregation tree T in the reply phase, denoted by E_{reply}.

$$E_{reply} = \frac{E_{reply}(r) + \sum_{i \in I_T} E_{reply}(i)}{|I_T| + 1} \tag{6}$$

III. Energy and Locality Aware Aggregation Tree for Skip Graph (ELATS)

In a Skip Graph, each node has two identifiers, a non-negative integer called numerical ID, and a binary string called name ID. In a Skip Graph with n nodes the search for name ID operation returns *address* the owns the most similar name ID to the search target by traversing $O(\log n)$. The similarity of a pair of name IDs is defined as the length of their common prefix. *ELATS* works on the top of a locality aware Skip Graph where the pairwise latencies of nodes correspond to the length of their name IDs' common prefix [7]. A longer prefix corresponds to the lower latency. In our view of an aggregation tree, all nodes in the Skip Graph are spanned by the aggregation tree. The node starts aggregation process by invoking *ELATS* is named the **initiator** which becomes the root of resulted aggregation tree. We define a **sub-domain** as the set of name IDs with a common prefix. The common prefix represents a sub-domain called its **sub-problem**. For example, in a Skip Graph with name ID size of 4 bits, corresponding sub-domain to the 01 sub-problem is $\{0100, 0101, 0110, 0111\}$.

ELATS is a recursive distributed algorithm where each recursion is executed on a single node. The node who executes a recursion of *ELATS* is called **executor**. *ELATS* is shown by Algorithm 1. As the inputs *ELATS* receives the address of the $executor$ node as a pointer, $subProblem$ which is a binary string defines the sub-problem, energy cost of current path denoted by E_c, latency of current path denoted by L_c, and the number of intermediate nodes on the current path denoted by $inernum$. The current path is defined as the path between the executor and root of aggregation tree. In *ELATS*, the latency of the current path is obtained by summing up the common prefix lengths of consecutive nodes on the path. Likewise, the energy cost current path corresponds to the average energy cost of non-leaf nodes on the path between the executor and root of aggregation tree.

ELATS first extracts the name ID of the executor denoted by $nameID$ (Algorithm 1, **line 1**). The corresponding sub-domain of the $subProblem$ denoted by $subDomain$ consists of the set of name IDs which start with the $subProblem$ prefix. All name IDs in $subDomain$ are as long as the executor's name ID denoted by $nameID$. Likewise, all name IDs in $subDomain$ start with $subProblem$ prefix of size $|subProblem|$ bits, and they differ only in their next $|nameID| - |subProblem|$ bits. Hence, size of $subDomain$

which corresponds to its number of name IDs is obtained as $2^{|nameID|-|subProblem|}$ (**line 2**). A $subDomain$ of size 1 represents the situation where $nameID$ is the only member of $subDomain$. In such a case, *ELATS* terminates since no parent-child relationship can be defined for one node (**lines 3 and 4**).

The core of *ELATS* is to decide on executor to be either the parent of all the $subDomain$ name IDs, or the parent at most half of them. We call the former as **covering all** case, and the latter as **covering half** case. In covering half case, only the name IDs have more than $|subProblem|$ bits common prefix with $nameID$ can become the children of executor. Moving from $|subProblem|$ to $|subProblem|+1$ ignores half of the $subDomain$ name IDs i.e., the ones who have exactly $|subProblem|$ bits common prefix. Since Skip Graph is locality aware, the remaining name IDs - which have at least $|subProblem|+1$ common prefix length with the $nameID$- constitute the expected lower latency half portion of $subDomain$. To make this decision, *ELATS* computes the affected energy cost and latency of current path affected by each of cases (**line 5**). The energy cost of current branch of aggregation tree in the covering all and covering half cases are illustrated by E_a and E_h, respectively. Each of these energy costs are obtained as the sum of the corresponding $E_{request}$ and E_{reply} obtained from Equations 3 and 6, respectively. In both equations $|I_T| = inernum$.

Equation 7 represents the latency of the current branch in the covering all case denoted by L_a. The latency of current branch is defined as the minimum common prefix length of $nameID$ with the name IDs in $subDomain$ which is equal to $|subProblem|$. This minimum common prefix corresponds to the maximum latency between the executor and nodes of $subDomain$. If the $executor$ becomes the parent of all $subDomain$ name IDs, all of the name IDs in $subDomain$ become leaves of the aggregation tree. The latency of the current branch is hence increased by the maximum latency between the executor and name IDs of $subDomain$ as shown in Equation 7. In the covering half case, since the minimum common prefix length of $nameID$ and the new sub-domain is $|subProblem|+1$, the corresponding latency denoted by L_h is obtained in a similar way shown by Equation 8.

$$L_a = L_c + |nameID| - |subProblem| \quad (7)$$

$$L_h = L_c + |nameID| - (|subProblem| + 1) \quad (8)$$

After computing the energy cost and latency of current branch of aggregation tree in both covering all and covering half cases, *ELATS* computes the latency and energy cost improvements of covering all compared to covering half cases denoted by imp_L and imp_E, respectively (**lines 6 and 7**). imp_L and imp_E are floating point values between 0 and 1, and provided by the $improvement$ function. On receiving two input arguments, the $improvement$ function divides their absolute value of difference over the greatest input value, and outputs the result.

A smaller than the e value of imp_L or imp_E conveys that going deeper than the current depth does not provide lower latency or better energy cost. This makes *ELATS* to choose

Algorithm 1: ELATS

Input: pointer $executor$, String $subProblem$, double E_c, integer L_c, int $inernum$

```
1  nameID = executor.getNameID();
2  subDomainSize = 2^|nameID|-|subProblem|;
3  if subDomainSize == 1 then
4  |  terminate;
5  computing L_a, L_h, E_a, E_h;
6  imp_L = improvement(L_a, L_h);
7  imp_E = improvement(E_a, E_h);
8  if imp_L < e or imp_E < e then
9  |  avnodes = extend(subDomain);
10 |  for i ∈ avnodes do
11 |  |  parent(i) = executor;
12 |  |  children(executor).add(i);
13 else
14 |  if commonBits(subProblem +'0', nameID) ==
   |   |subProblem| then
15 |  |  α = searchByNameID(subProblem +'0');
16 |  |  if α ≠ NULL then
17 |  |  |  parent(α) = executor;
18 |  |  |  children(executor).add(α);
19 |  |  |  α.ELATS(α, subProblem +'0', E_h , L_h,
   |  |  |   inernum + 1);
20 |  |  ELATS(executor, subProblem +'1', E_h , L_h,
   |  |   inernum + 1);
21 |  else
22 |  |  α = searchByNameID(subProblem +'1');
23 |  |  if α ≠ NULL then
24 |  |  |  parent(α) = executor;
25 |  |  |  children(executor).add(α);
26 |  |  |  α.ELATS(α, subProblem +'1', E_h, L_h,
   |  |  |   inernum);
27 |  |  ELATS(executor, subProblem +'0', E_h , L_h,
   |  |   inernum);
```

the covering all case. e is a close to zero local constant of executor determined based on its properties like bandwidth and computational power. To perform covering all case, *ELATS* first finds all available name IDs in $subDomain$. A name ID is available if it has been assigned to an existing node in the Skip Graph. To find the available name IDs, *ELATS* employs the $extend$ function which applies a *searchForNameID* for each of the name IDs in $subDomain$, and checks their existence. The set of all available nodes of $subDomain$ is denoted by $avnodes$. *ELATS* adds each node in $avnodes$ to the children set of the executor, as well as makes the executor as the parent of all nodes in $avnodes$ (**lines 8-12**).

Large improvements in both imp_L and imp_E promise to meet better energy cost and lower latency as going deeper. In this case, the executor node needs to divide the $subDomain$ into two smaller sub-domains and choose covering half case (**line 13**). $nameID$ starts with the $subProblem$ prefix of size $|subProblem|$. The $|subProblem|+1^{th}$ bit of $nameID$ is either 0 or 1. If the $|subProblem|+1^{th}$ bit of $nameID$ is equal to 1, the common prefix length of $subProblem+'0'$ and $nameID$ computed by $commonBits$ function is equal to $|subProblem|$ (**line 14**). For example, if $nameID = 01010011000$ and $subProblem = 0101001$, then $subProblem+'0' = 01010010$, and the common prefix of $subProblem+'0'$ and $nameID$ is 0101001 which is equal

to the $subProblem$ itself. In this case, the executor node should deliver the aggregation tree covering the sub-domain corresponds to $subProblem +' 0'$ to another node which has the name ID prefix of $subProblem +' 0'$. The executor node performs a $searchForNameID$ to find address of node α has the $subProblem +' 0'$ prefix in its name ID (**line 15**).

If node α exists, the executor node introduces itself as the parent of α and adds α to its children set (Algorithm 1, lines **16-18**). Likewise, *ELATS* calls a fresh recursion of itself on the node α for the $subProblem +' 0'$. In the fresh recursion of *ELATS*, the current average energy cost of intermediate nodes on the path between the root of aggregation tree and node α is equal to the already computed E_h. Likewise, the latency of path from root node to α is equal to the already computed L_h. Since the executor is the parent of α in the aggregation tree, the number of intermediate nodes on the path between the root node and node α is one more than the number of intermediate nodes on the path between the root node and executor (**line 19**).

ELATS also calls a fresh recursion on the executor node for the $subProblem +' 1'$ and with the similar average energy cost and latency arguments. However, since executor node runs the fresh recursion, no intermediate node is added (**line 20**). **Lines 21-27** show the similar procedure for the case where $|subProblem| + 1^{th}$ bit of the $nameID$ is equal to 0. Assuming that the initiator of aggregation tree is node β, the initial call to *ELATS* is β.*ELATS*$(\beta, NULL, 0, 0, 1)$.

Time Complexity: As shown by lines 2-4 of Algorithm 1, a sub-problem that is as long as the name ID size of Skip Graph terminates the recursion of *ELATS*. The total number *ELATS*'s recursions on a single node is therefore bounded by the name ID size of Skip Graph. In a Skip Graph with n nodes, the name ID size is $\lceil \log n \rceil$. Hence, *ELATS* enforces $O(\log n)$ recursions on each node of the Skip Graph. The recursions of *ELATS* are executed in parallel on different nodes.

IV. Related Works

Table I provides a comparison between the decentralized aggregation tree schemes and our proposed *ELATS*. A prior work [21] proposes a parent function for the circular namespaces like Chord [9] where each node covers all its predecessors. Given a node's identifier, the parent function returns back the parent's identifier for that node. However, the parent function is not applicable to the Skip Graph which does not follow a circular continuous namespace.

Applying a distributed breadth first search (BFS) [24] on the network's graph results in a spanning BFS-tree which can be used as an aggregation tree. A BFS-tree guarantees the minimum number of intermediate nodes between each pair of nodes which implicitly results in a shorter aggregation tree. A short aggregation tree helps saving energy by reducing the number of exchanged messages. However, the locality awareness of the resulted aggregation tree is completely ignored in the BFS-tree.

Strategy	Energy Aware	Locality Aware
Parent Function [21]	No	Yes
BFS-trees [24]	Yes	No
SP-trees [25]	No	Yes
Broadcast Tree [2], [21], [26], [27]	Yes	No
Prefix-based Tree [26], [29], [30]	No	Yes
ELATS	**Yes**	**Yes**

TABLE I: Comparison of decentralized aggregation trees

An aggregation tree has two degrees of freedom: nodes' degree, and tree's height. [25] claims the average nodes' degree in a shortest path-based (SP-based) tree drops as the system size scales up, and converges to the degree of 3. Also, the degree of a node in this approach does not go beyond 8. An SP-tree provides locality awareness. However, limiting nodes' degree makes the aggregation tree to grow in height as the system size scales up. This results in increasing number of exchanged messages during the aggregation which negatively affects the energy cost of the tree. In a spanning **broadcast tree** [2], [21], [26], [27] each node broadcasts the received message to all its neighbors. The broadcasting procedure continues in a top-down manner until the message reaches the leaves. A broadcast tree can be used as an aggregation tree by broadcasting the aggregation request from the root node. Each node broadcasts the request to its children, aggregates and replies their responses back to its parent.

Similar to Skip Graph, Kademlia [28] is a prefix-based routing DHT. There are many **prefix-based** broadcast trees [26], [29], [30] proposed for Kademlia. In a prefix-based broadcast tree, the system is divided into disjoint regions, where each region covers a specific prefix. A broadcast initiator initiates the broadcast by contacting one node per region.

V. Simulation Results

We extended the Skip Graph simulator SkipSim [22] and developed models to simulate the aggregation tree algorithms. We implemented *ELATS*, broadcast tree, and prefix-based tree in SkipSim, and compared their performance in terms of energy and locality awareness. Each simulation consists of 100 random topologies. In our simulations, we consider $E = 8J$, $U = 1J$, and for *ELATS* $e = 0.01$.

Figure 1 shows the average energy cost of non-leaf nodes of the aggregation tree as the system capacity scales up. The system capacity is defined as the maximum number of registered users to the system. As the system capacity scales up exponentially the average number of children assigned to non-leaf nodes in prefix-based aggregation tree increases exponentially. This results in an exponentially increasing energy cost on non-leaf nodes of the prefix-based tree. The degree of each node in the broadcast tree is bounded by the number of its neighbors. Having a system capacity of n, the number of neighbors of a node in the Skip Graph is $O(\log n)$. Hence the exponentially increasing system capacity does not have a noticeable effect on the energy cost of the broadcast tree. Similar to the broadcast tree, *ELATS* acts independent of the system capacity in terms of energy cost and provides the similar energy cost as the broadcast tree which is the most energy efficient existing solution.

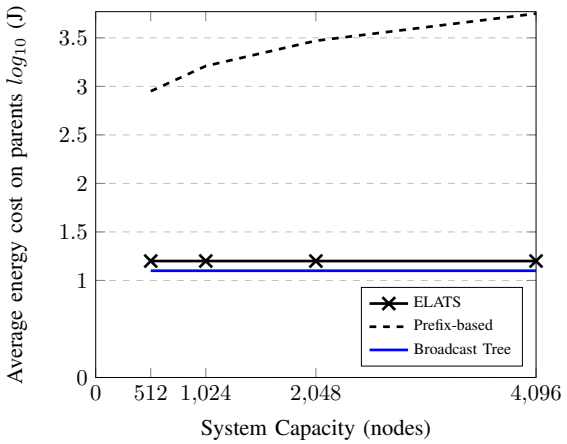

Fig. 1: Average energy cost on parent nodes in the aggregation tree vs System capacity. Y-axis shows the base 10 logarithms of the average energy cost.

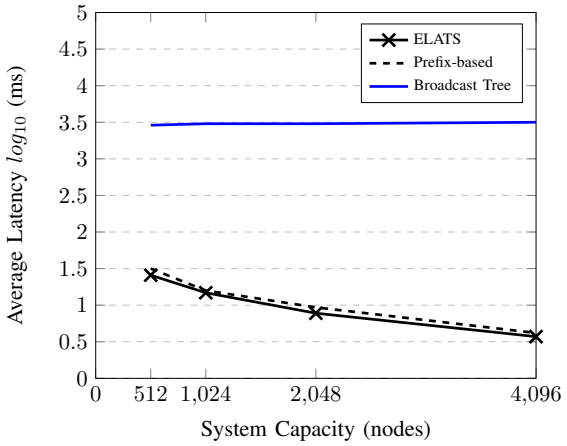

Fig. 2: The average of maximum root to leaves' path latency in the aggregation tree vs System size. Y-axis shows the base 10 logarithms of average path latency.

Figure 2 shows the average of the maximum latency of root to leaves' path in aggregation trees as the system capacity scales up. In a locality aware Skip Graph as the system capacity increases, the system is divided into more condensed regions and more accurate locality aware name IDs are achieved [7]. This results in an improvement in the latencies of prefix-based and *ELATS*. Since the broadcast tree, which performs as the best energy efficient existing solution, is not locality aware, it performs the worst in terms of latency when compared to *ELATS* and prefix-based tree. As shown in Figure 2, in comparison to the prefix-based which acts as the best existing locality aware solution, *ELATS* improves the average value of the maximum root to leaves' path latency with the gain of about **8%**.

VI. Conclusion

To improve the energy efficiency and response time of the aggregation procedure in a Skip Graph, we propose *ELATS*, the first fully decentralized energy and locality aware aggregation tree. We simulated and compared *ELATS* with state-of-the-art aggregation tree algorithms for DHTs. Compared to the best existing solutions which are either locality aware or energy aware, *ELATS* improves aggregation latency with a gain of about **8%** while simultaneously providing similar energy efficiency to the best energy aware existing solution.

References

[1] J. Aspnes and G. Shah, "Skip graphs," *ACM TALG*, 2007.

[2] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient broadcast in structured p2p networks," in *International workshop on Peer-to-Peer systems*. Springer, 2003.

[3] E. Udoh, *Cloud, grid and high performance computing: emerging applications*. Information Science Reference, 2011.

[4] S. Batra and A. Singh, "A short survey of advantages and applications of skip graphs," *IJSCE*, 2013.

[5] T. Shabeera, P. Chandran, and S. Kumar, "Authenticated and persistent skip graph: a data structure for cloud based data-centric applications," in *ACM CSS 2012*.

[6] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "Awake: decentralized and availability aware replication for p2p cloud storage," in *IEEE SmartCloud, 2016*.

[7] ——, "Locality aware skip graph," in *IEEE ICDCSW, 2015*.

[8] ——, "Laras: Locality aware replication algorithm for the skip graph," in *IEEE NOMS 2016*.

[9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, 2001.

[10] S. Taheri-Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Security and privacy of distributed online social networks," in *IEEE ICDCSW, 2015*.

[11] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of P2P systems*, 2003.

[12] T. Jurdziński, M. Kutyłowski, and J. Zatopiański, "Energy-efficient size approximation of radio networks with no collision detection," *Computing and Combinatorics*, 2002.

[13] R. Zhou and K. Hwang, "Gossip-based reputation aggregation for unstructured peer-to-peer networks," in *2007 IEEE IPDPS*.

[14] M. Bagaa, A. Derhab, N. Lasla, A. Ouadjaout, and N. Badache, "Semi-structured and unstructured data aggregation scheduling in wireless sensor networks," in *INFOCOM, 2012 Proceedings IEEE*.

[15] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*.

[16] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM TOCS*, 2005.

[17] M. Haridasan and R. Van Renesse, "Gossip-based distribution estimation in peer-to-peer networks." in *IPTPS*. Citeseer, 2008.

[18] N. Bisnik and A. Abouzeid, "Modeling and analysis of random walk search algorithms in p2p networks," in *IEEE IPTPS 2005*.

[19] R. Zhou and K. Hwang, "Trust overlay networks for global reputation aggregation in p2p grid computing," in *IPDPS*. IEEE, 2006.

[20] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks: algorithms and evaluation," *Performance Evaluation*, 2006.

[21] J. Li, K. Sollins, and D.-Y. Lim, "Implementing aggregation and broadcast over distributed hash tables," *ACM SIGCOMM*, 2005.

[22] "Skipsim: https://github.com/yaahaanaa/skipsim."

[23] Ö. Özkasap, E. Cem, S. E. Cebeci, and T. Koc, "Flat and hierarchical epidemics in p2p systems: Energy cost models and analysis," *Future Generation Computer Systems*, vol. 36, pp. 257–266, 2014.

[24] M. Dam and R. Stadler, "A generic protocol for network state aggregation," 2005.

[25] D. Dolev, O. Mokryn, and Y. Shavitt, "On multicast trees: structure and size estimation," *IEEE/ACM Transactions on Networking*, 2006.

[26] A. D. Peris, J. M. Hernández, and E. Huedo, "Evaluation of alternatives for the broadcast operation in kademlia under churn," *Peer-to-Peer Networking and Applications*, 2016.

[27] K. Huang and D. Zhang, "Dht-based lightweight broadcast algorithms in large-scale computing infrastructures," *Future Generation Computer Systems*, 2010.

[28] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002.

[29] M. Wählisch, T. C. Schmidt, and G. Wittenburg, "Broadcasting in prefix space: P2p data dissemination with predictable performance," in *IEEE ICIW 2009*.

[30] A. D. Peris, J. M. Hern'ndez, and E. Huedo, "Evaluation of the broadcast operation in kademlia," in *2012 IEEE HPCC-ICESS, 2012*.